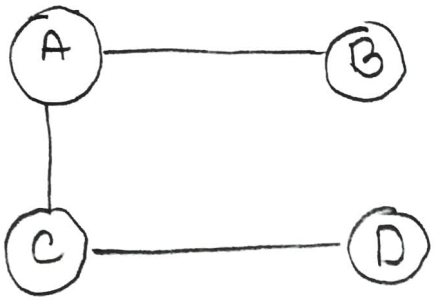
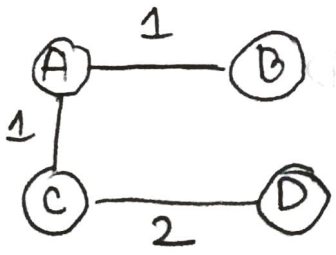
Assignment 2

Problem 1

First, lets assume a graph with four Nodes like this,

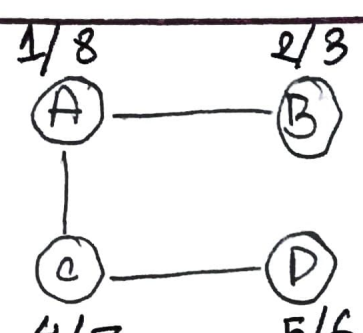


Its an undirected graph, and if we run bfs over if it if will first traverse A to B and C, ~~then~~ then C to D



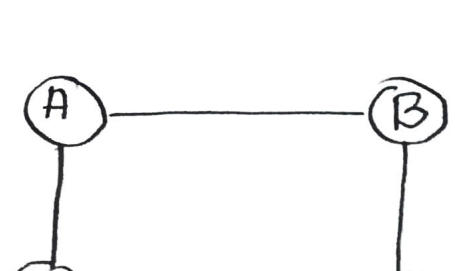
ordered list : A B C D

In this same graph if we run DFS it will start from A then B, then C and D

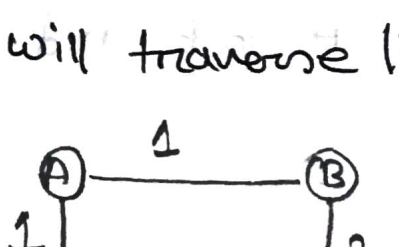


ordered list : ABCD

Lets consider another undirected graph:

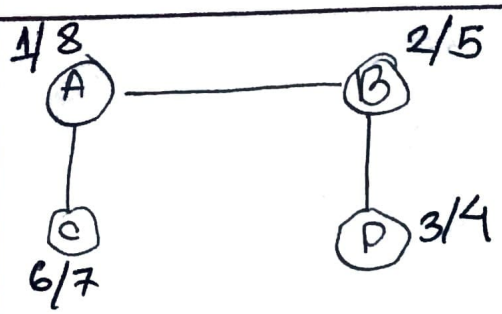


If we run BFS over it the following graph will traverse like this



order : ABCD

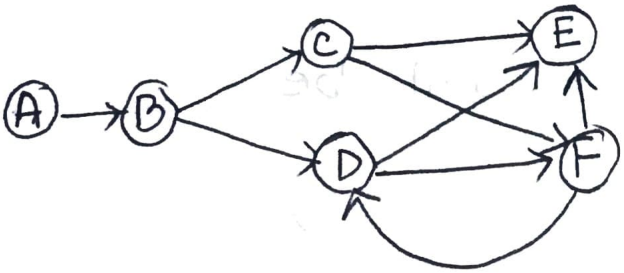
Now if we run DFS in the same graph the solution is diffrent now,



order : A B D C

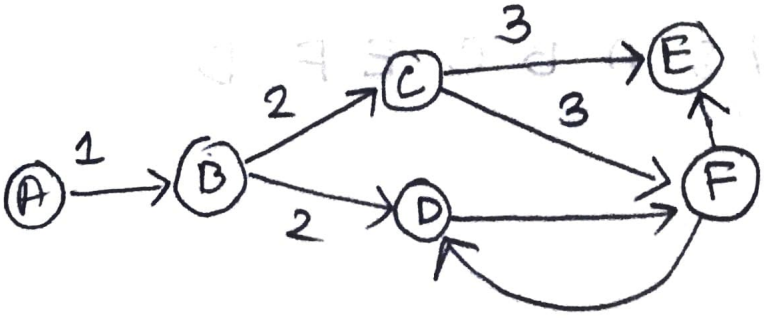
In the first graph we get same answer for bfs, dfs. But in the second one the order is different from each other.

Problem 2

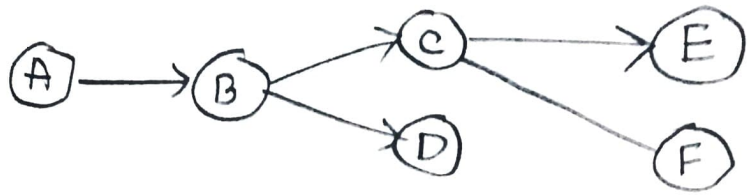


For BFS

if we run BFS over it will look like this

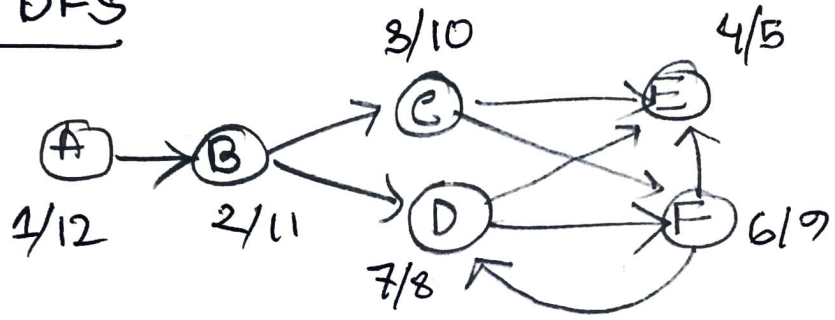


∴ the tree and list will be,

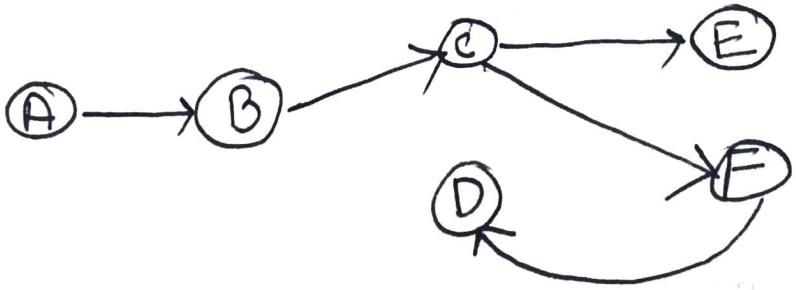


ordered List: A B C D E F

For DFS



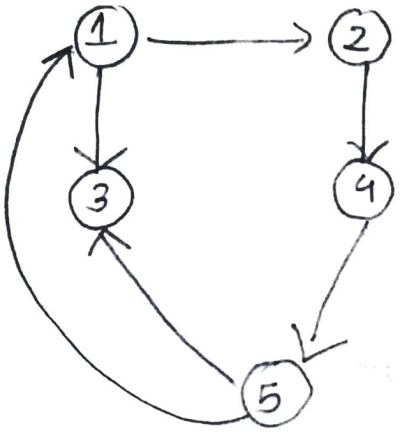
∴ the tree and list will be



ordered List: A B C E F D

Problem 3

Lets consider a graph,

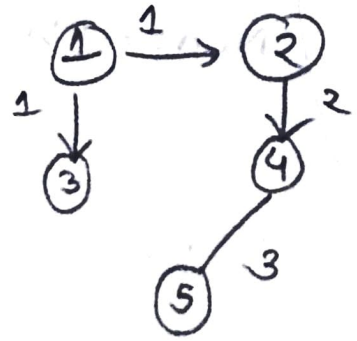


* The graph is directed
* Each Node has both Incoming and outgoing edges.
* here $n = 5$ and $m = 6$

Now I will Implement BFS on it. Because it makes sure that each node can traverse to any Node both in directed and undirected graph

Now, Lets implement BFS in the above graph and see,

First if the source is "1" then the order will be



order: 1,2,3,4,5

Now for the other source 2, 3, 4, 5 we get different order

Source 2 → 24513
Source 3 → 32451
Source 4 → 45132
Source 5 — 51324

Here we can see the orders are different. the similarity between this orders is that, they visit all the Nodes from the source Node.

Time complexity of BFS,

```
BFS(graph, Node, endpoint, visited):
    Do visited
    Do queue
    while queue is not empty:          } = Number of Nodes (n)
        Do m ← pop
        if m = end
            break
    for each neighbour (edge):         } number of edges (m)
        Do visited
        Do Queue (neighbour)
```

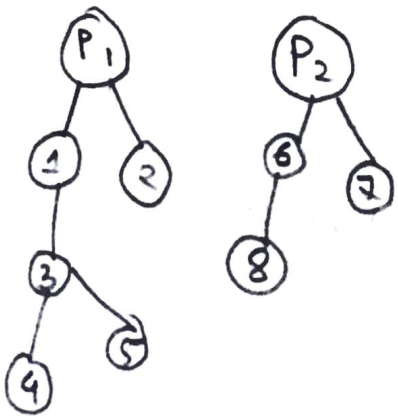
In the above pseudo code, the while loop will run for, 'n times', where n is the number of Nodes, and the for loop will run for number edges time for each Node [if we use adjacency list]

∴ the time complexity : ~~O[n+m]~~

$O(n+m)$

Problem – 4

Lets assume there are $n = 2$ power plants and $n^3 = 8$ buildings



here we can have disconnected graph.

And as In question the said each building can be recursively powered we know that ~~DFS~~ In DFS we visit each vertex Recursively. so here to determine this kind of scenario we can use DFS.

To install a generator we have to check in which powerplant have the most buildings.

we have to modify the dfs code,
we have to assign a count varible, for each powerpoint the count varible must increase and assign in a emply list

```
visited  F]
List []
count 0
DFS visit (g, node)
    for each node ( )
        if not in visited
            ~~lbt.append~~
            DFS visit (g, node)
            count ++
        else:
            list.append (count)
            count = 0
DFS (g, end)
    For each node in graph
        if not in visited
            DFS vist (g, node)
```

then we have to traverse the "List" and check out for the maximum number. And the (index+1) no is the power point we want.

time complexity:

In DFS visit, the for loop will run for number of vertex time, so it will run for n^3 times

In worst case one of the power point will have all buildings

so then the loop in the DFS will also run for n^3 times.

as in DFS visit the time complexity we have

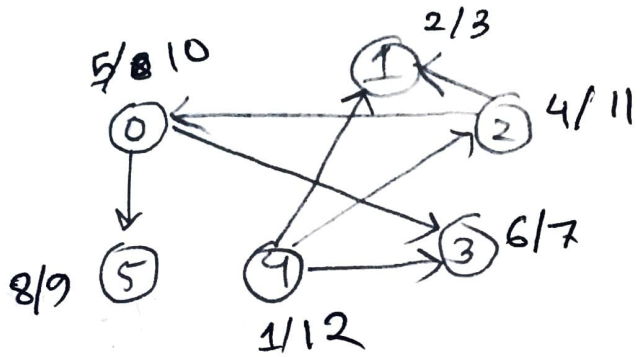
$$O(n^3 * n^3)$$

$$= O(n^6)$$

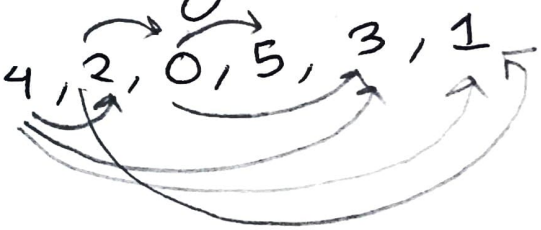
Problem 5

a

Here first we start from Node "4",

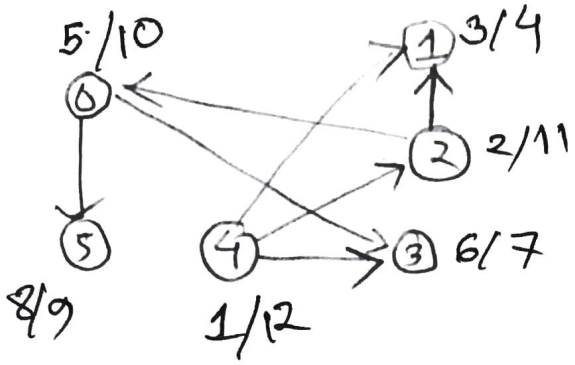


The order after sorting with finishing time :



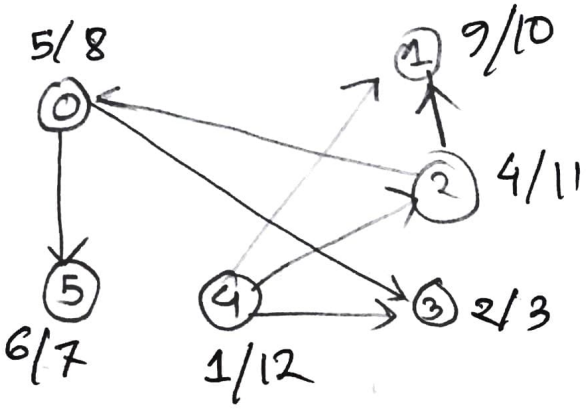
here we can go 1, 2, 3 after 4, then after 2 0 and 1 can be done, after 0, 5 and 3 can be done.

Now there can be other orders too, if we still start from 4.



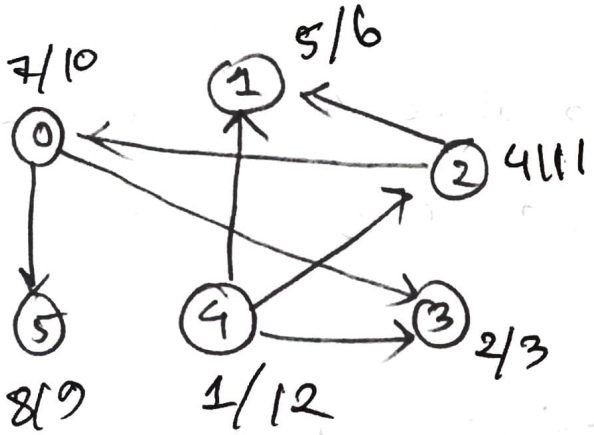
order: 4 2 0 5 3 1

Same as previous one if we first visit 2 after 4



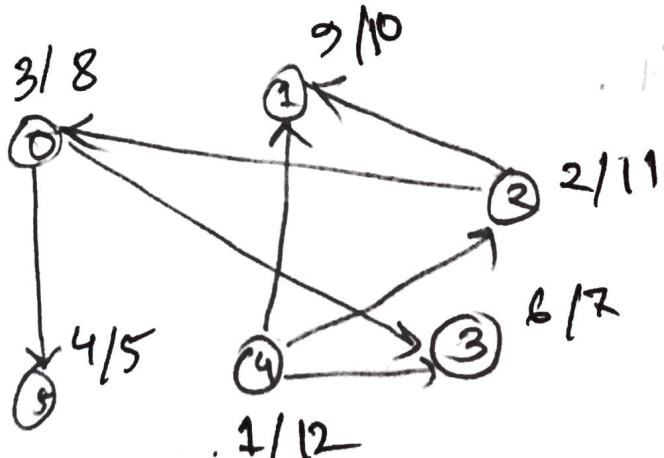
order : 4 2 1 0 5 3

here if we visit 3 after 4 and, 0 after 2 this is the order we get.



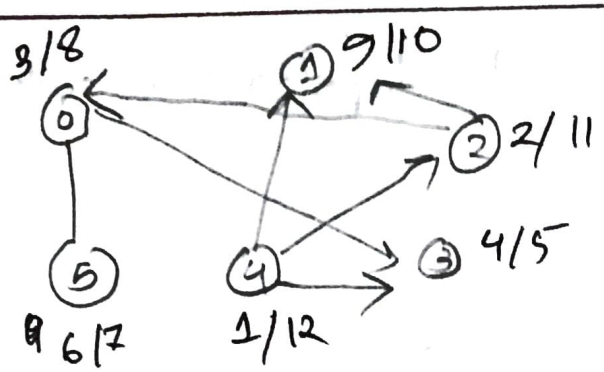
order: 4 2 0 5 1 3

here after 4 I visit 3 and, after 2 I visit 1.



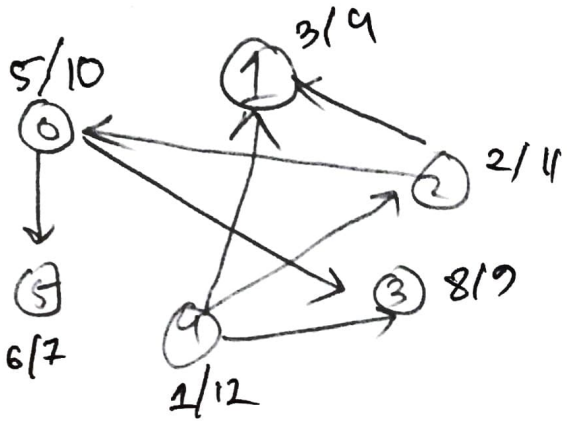
order: 4 2 1 0 3 5

here after 4 I visit 2 then 0 then 3 then 5 and then 1

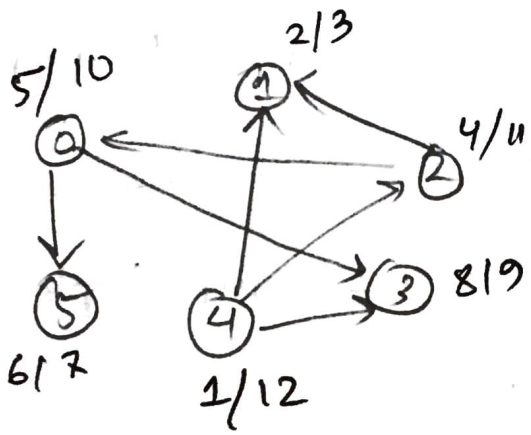


order: 4 2 1 0 5 3

here after 4 I visit 2 then 0 then 3 then 5 and then 1.

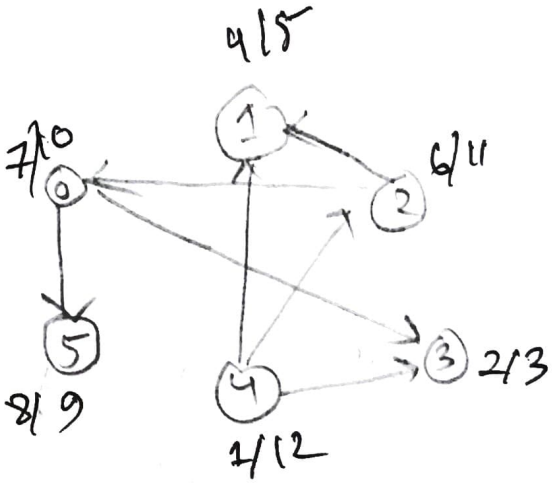


order: 4 2 0 3 5 1



on-der: 4 2 0 3 5 1

here after 4 I visited 1 then 2 then 0 then 5 then 3 ~~then~~



order: 4 2 0 5 1 3

here after 4 I visited 3 then 1 then 2 then 0 and then 5.

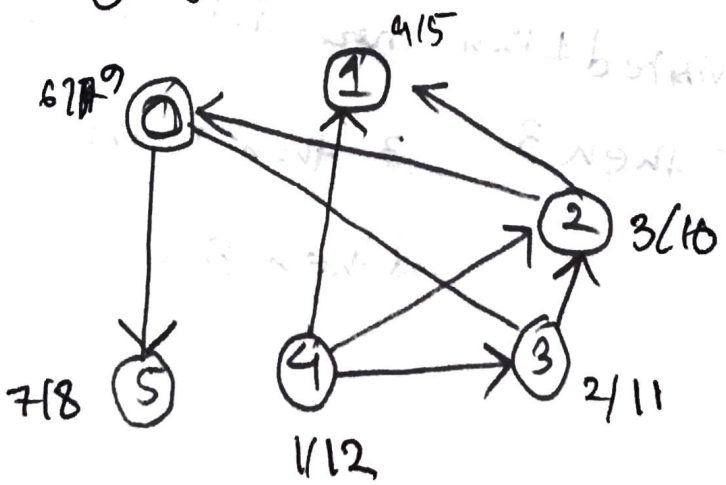
so we got 5 distinct topological ordering

=> 4 2 0 5 3 1
4 2 1 0 5 3
4 2 0 5 1 3 Total: 5
4 2 1 0 3 5
4 2 0 3 5 1

b

For maintaining topological order: A graph has to be directed Acyclic. If there is a cycle in a graph then we cant implement topological sort in it.

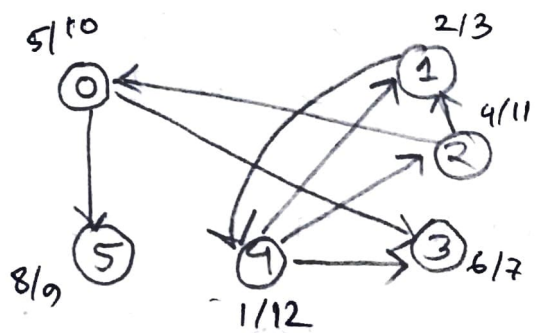
if we consider a single edge from 3→2 in the following graph



So the order will be : 4, 3, 2, 0, 5, 1

here we can see we can go from 4 to 1,2,3, we can go 3 to 2, we can go 2 to 0, 1, but according to the graph we cant visit 0 to 3 it would be a backward traverse. so the topological order does not exist

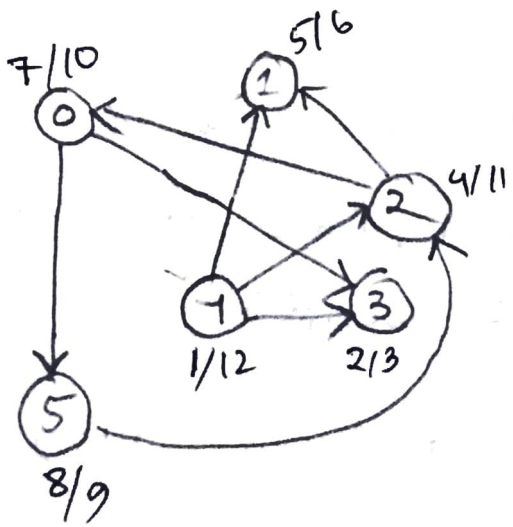
Similarly there can be 8 more single directed edges can be added in the graph with no topological ordering.



New edge = 1 → 4

order : 4, 2, 0, 5, 3, 1

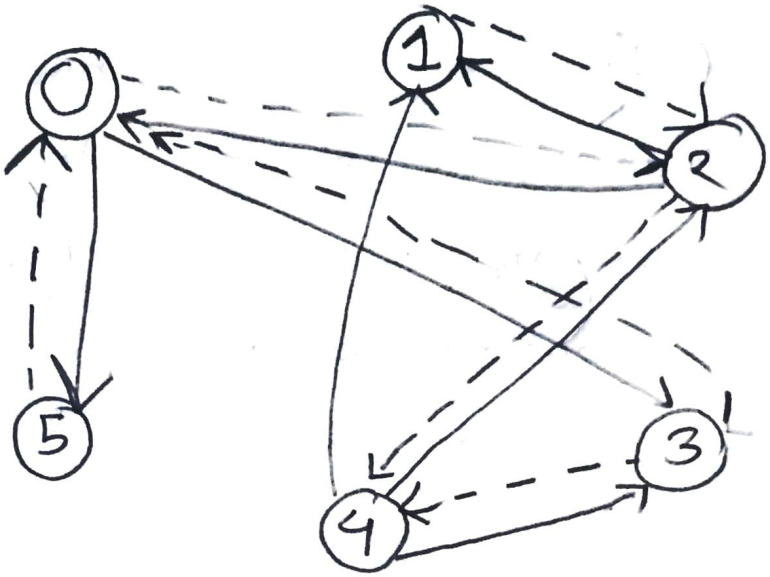
* cant visit 1 to 4



New edge : 5 → 2

order : 4, 2, 0, 5, 1, 3

* can not visit 5 to 2, backward direction.



So this 5 dolted edges : $3 \rightarrow 4$

$2 \rightarrow 4$

$3 \rightarrow 0$

$5 \rightarrow 0$

$0 \rightarrow 2$

By taking these individual ~~steps~~ edges, if we try to sort topologically, the sort doesn't order correctly

like in case of $3 \rightarrow 4$, the order is 420513

here we cant visit $3 \rightarrow 4$, like all the other above edges dont work.

∴ Total number of distinct single edges are: 9

above 9 edges can be added to the graph with no topological ordering.